Classification of the Gamma dataset

by Elisha Gretton

Submission date: 20-Dec-2021 11:34AM (UTC+0000)
Submission ID: 167012251
File name: MAS61007-180188196-Assignment2.pdf (195.58K)
Word count: 1673
Character count: 8122

Classification 180188196

Introduction

In this assignment, we will be looking at the gamma dataset. The goal is to classify particles, which come from gamma or hadron particles using machine learning techniques. There are 11 variables in the data: Length, Width, Size, Conc, Conc1, Asym, M3Long, M3Trans, Alpha, Dist, and class.

Section 1: Logistic Regression

Logistic regression aims to fit a generalized linear model using maximum likelihood techniques, using the equation $p(E(y)) = \alpha + \beta_1 x_1 + \beta_1 x_2$.

When running this technique on the whole dataset, we get an accuracy of 77.694%. The estimated logistic regression is as follows,

$$p(E(y)) = -0.315 - 1.5585Length - 0.0248Width - 0.2669Size + 0.1423Conc - 0.7681Conc1$$

$$+ 0.0052Asym + 0.4996M3Long + 0.0738M3Trans - 1.2965Alpha - 0.0453Dist.$$

It looks like `Length`, `Conc1`, `Long`, and `Alpha` all have large coefficients (when taking absolute values), therefore we can infer that these are important features for classifying gamma particles.

When doing some validation and splitting the data into a training and test set with a split of 80:20, an accuracy of 79.5% is achieved. For this regression, we get the most influential coefficients for the same features, with `Length` and `Alpha` the most important of them all.

Section 2: Decision Trees

The data has been split up into a training, validation, and testing set with an 80:10:10 split. The same sets will be used throughout the rest of the report. Default settings of a classification decision tree give validation accuracy of 0.79 and test accuracy of 0.805.

By tuning a decision tree based on `max_depth`, the overall accuracy of the tree increases. A validation accuracy of 0.8325 and test accuracy of 0.855 are achieved for a `max_depth` of 9. These were the highest accuracies, therefore a `max_depth` of 9 is chosen. If we look at the top two layers of the tree, `Alpha` and `Length` are the most important features. The data is initially split using `Alpha(<=19.56)`, and then into `Length(<=112.968)` and `Length(<=41.224)`.

Section 3: Random forests

After this, we will be looking at random forests and performing some parameter tuning to optimize the model. Under default parameters, the random forest model achieved a validation accuracy of 0.8575 and a test accuracy of 0.8675.

The parameter `n_estimators` is investigated. When `n_estimators=21`, the highest validation accuracy (0.8725) and test accuracy (0.8725) were achieved. This value is taken for `n_estimators`, and then the parameter `max_depth` is tuned. A `max_depth` of 6 gave a validation accuracy of 0.8275 and the highest test accuracy of 0.875. From tuning these two parameters, the optimal parameters are `n_estimators=21` and `max_depth=6`.

Some other parameters that were taken into consideration were `min_samples_split` and `min_samples_leaf`. The `min_samples_split` is the minimum number of samples required to split an internal node. The `min_samples_leaf` is the minimum number of samples required to be at a leaf node. From increasing `min_samples_leaf`, the validation and test accuracy decreases, therefore the default `min_samples_leaf=2` is taken. Next, `min_samples_split` is tuned. From increasing `min_samples_split` from 2 to 3, test accuracy increases from 0.84 to 0.875. This means our tuned parameters are taken as `n_estimators=21, max_depth=6` and `min_samples_split=3,` giving a test accuracy of 0.875.

Section 4: Boosting

Boosting is used in classification problems to overcome the issue of bias-variance tradeoff. This technique tackles that by giving correctly predicted outcomes a lower weight and misclassified outcomes a higher weight to improve prediction accuracy. There are many parameters to tune when using boosting. For example, there are still the usual tree parameters such as `max_depth`, however, also boosting parameters such as `learning_rate`.

Using the default model with a `learning_rate` of 0.1, a validation accuracy of 0.8725, and a test accuracy of 0.865 is achieved. Tree parameters such as `max_depth` and `n_estimators` are tuned to begin with. When tuning `n_estimators`, a value of 300 gave a validation accuracy of 0.8675 and a test accuracy of 0.885. This parameter is set to 300, and then `max_depth` is tuned. A `max_depth` of 5 achieved a validation accuracy of 0.87 and a test accuracy of 0.8875. The optimal tree parameters are `n_estimators=300` and `max_depth=5`.

Next, the boosting parameters are investigated such as `learning_rate` and `subsample`. When the `learning_rate` increases from 0.1 to 0.2, test accuracy decreases to 0.875, therefore a `learning_rate=0.1` is used. `Subsample` was changed from 1 to 0.8, which gave a validation accuracy of 0.87 and test accuracy of 0.8725.. This test accuracy is lower than when `subsample` is default at 1, which gives a test accuracy of 0.8875. Therefore the optimal parameters for a gradient boosting classifier is `learning_rate=0.1, subsample=1, n_estimators=300` and `max_depth=5,` giving a test accuracy of 0.8875.

Section 5: Neural Networks

Neural networks are another technique that is inspired by the decision-making of the human brain. Artificial functions are created in a network to classify data, and then parameters are tuned to optimize these networks. A neural network of 12 input nodes to a hidden layer of 8 nodes, and an output of 1, is used as a default model.

To start, `optimizer=rmsprop, loss=binary_crossentropy, epochs=50, batch_size=10`. The results give a training loss of 0.3214 and training accuracy of 0.8615. The test accuracy is 0.8625. Below are two graphs plotting accuracy/loss vs epochs.

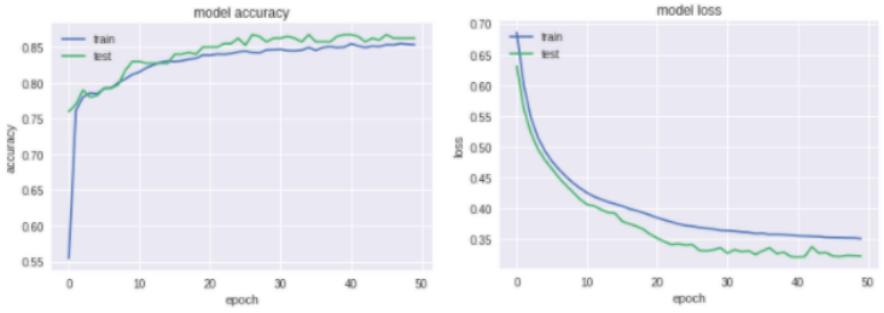

Figure 1: Accuracy Vs. Epochs graph (left), Loss Vs. Epochs graph (right).
Parameters: optimizer=rmsprop, loss=binary_crossentropy, epochs=50, batch_size=10.

From the graph on the left, it looks like training accuracy starts to converge around `epochs=20`. As a result, `epochs` are reduced. The testing accuracy roughly follows the curve for the training accuracy, however after one epoch, the testing accuracy shoots up to 0.76. A reason for this could be by looking at the training set. Around `epochs=1`, the training accuracy shoots up to 0.76, inferring overfitting and could be the reason why our test accuracy starts so high. On the graph on the right, the training and testing loss are similar and do decrease as desired. However, we must consider overfitting and further tune the model.

When `epochs=20`, training accuracy is 0.8387 and the testing accuracy is 0.8575. In Figure 2, another Accuracy Vs. Epochs graph for `epochs=20` is plotted.

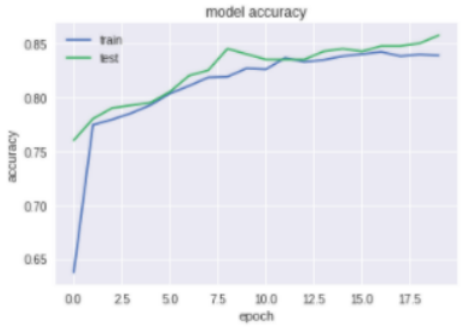

Figure 2: Accuracy Vs. Epochs graph
Parameters: optimizer=rmsprop, loss=binary_crossentropy, *epochs=20*, batch_size=10.

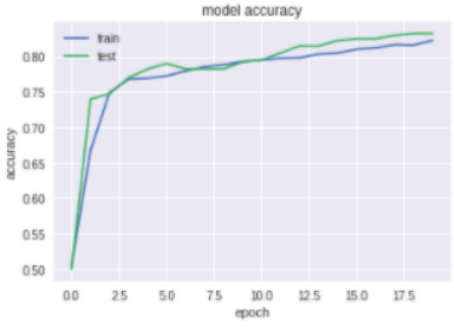

Figure 3: Accuracy Vs. Epochs graph
Parameters: optimizer=rmsprop, loss=binary_crossentropy, *epochs=20*, *batch_size=10*.

Once again, the testing accuracy starts high at around 0.76, which could be due to overfitting. There are also fluctuations in our accuracy, inferring our model is getting lucky on some epochs and can still be more finely tuned.

Next, `batch_size` was increased as the training process takes longer when `batch_size` is small. A `batch_size` of 40 is used, which increased training accuracy to 0.8228 and

testing accuracy to 0.8325. Moreover, the Accuracy Vs. Epochs curve is a lot smoother (Figure 3), which looks like this model is a better fit.

Other optimizers were used, such as Adam and AdaGrad. Adam achieved a higher test accuracy of 0.8425, and AdaGrad achieved a disappointing test accuracy of 0.4875. Adam also produces a smooth increase in accuracy, therefore is taken as the best optimizer.

Moreover, the learning rate of Adam was increased from 0.001 to 0.01. This increased the test accuracy from 0.8425 to 0.8725. Despite this, this resulted in a lot of fluctuations in the accuracy and loss, which infers the model is overfitting. A smaller increase in learning rate e.g. 0.002 was tested, however, this still caused some jaggedness in the Accuracy Vs. Epochs graph, suggesting overfitting. Therefore, the optimal parameters for this neural network are `epochs=20`, `batch_size=40`, `optimizer=Adam`, `learning_rate=0.001`, and `loss=binary_crossentropy` with a test accuracy of 0.8475.

Conclusion *(and Personal Note)*

All in all, all methods reach a high training and testing accuracy. However, it is the gradient boosting model with parameters `learning_rate=0.1, subsample=1, n_estimators=300,` and `max_depth=5,` that achieves the highest test accuracy of 0.8875. Testing this with another training and testing set, the highest test accuracy of 0.87 is achieved with this particular gradient boosting model, confirming this is the best model for the data.

If I were to continue this investigation, I would look at further tuning the neural network by changing the architecture and testing more values of `batch_size` and `epochs` on different optimizers. I would also want to cross-validate the neural network using the validation set. Additionally, I would look at creating accuracy curves for the other models, especially Boosting, to look for any overfitting. It would also be nice to automate the parameter tuning process so I could test all my models with lots of different training and testing sets, rather than running the code separately as this took a lot of time. I would also look at other parameters as I focused a lot on `max_depth` and `n_estimators`, and maybe look up what are suitable estimations of these parameters. For example, in the decision tree model, I only tested `n_estimators` from 0 to 30, but in boosting I tested `n_estimators` 50 to 400 from a tutorial I saw online. This is a bit inconsistent so I would need to look up what are appropriate estimations of these parameters given the model.